[序 2](#_Toc2770)

[修改网络配置 2](#_Toc8757)

[使用yum进行软件包管理： 3](#_Toc24904)

[安装ifconfig 3](#_Toc15121)

[安装ssh 3](#_Toc13470)

[安装lrzsz通信套件 3](#_Toc3945)

[安装python基础环境和python 3](#_Toc21532)

[VIM 4](#_Toc17529)

[安装图形桌面 4](#_Toc16116)

[安装mongodb 5](#_Toc6144)

[方式一：源码安装方式 5](#_Toc29141)

[方式二：yum包管理工具安装方式 5](#_Toc14874)

[Request 6](#_Toc4591)

[request模块支持的http方法 6](#_Toc8883)

[url构造方法 7](#_Toc26898)

[常见状态码 7](#_Toc10147)

[重定义请求头 8](#_Toc981)

[设置超时时间 8](#_Toc192)

[设置cookies 8](#_Toc2536)

[保持登录session 9](#_Toc4455)

[设置代理 9](#_Toc27130)

[证书校验 9](#_Toc19264)

[re模块 10](#_Toc32225)

[complie函数 10](#_Toc23637)

[match方法 10](#_Toc12513)

[search方法 10](#_Toc10932)

[findall方法 11](#_Toc16708)

[split方法 11](#_Toc29226)

[sub方法 11](#_Toc16293)

[xpath语法 12](#_Toc15585)

[BeautifulSoup4 13](#_Toc16791)

[方法选择器 14](#_Toc30181)

[css选择器 14](#_Toc10903)

[lxml 14](#_Toc3294)

[案例：lxml爬取51招聘网 14](#_Toc17763)

[爬取数据并保存到mongo 15](#_Toc7291)

[案例：爬取51招聘网全部网页信息 16](#_Toc31017)

[handle\_51job.py 16](#_Toc20843)

[handle\_mongo.py 19](#_Toc311)

[设置代理 19](#_Toc32205)

[实操案例爬取豆瓣T250 20](#_Toc779)

[crawl\_douban\_movie\_info\_top250.py 20](#_Toc25988)

[handle\_mongo.py 21](#_Toc32239)

[PyMongo库的使用 22](#_Toc29143)

[报错 23](#_Toc10387)

[selenium模块 23](#_Toc12086)

[selenium元素定位技巧 25](#_Toc13366)

[显示等待 25](#_Toc20010)

[隐式等待 25](#_Toc27234)

[案例：Selenium\_51job 26](#_Toc26802)

[报错 27](#_Toc330)

[scrapy 框架 27](#_Toc6044)

[items.py 28](#_Toc9285)

[tubatu.py 28](#_Toc32672)

[pipelines.py 29](#_Toc22784)

[middlewares.py 30](#_Toc31806)

[设置随机请求头 30](#_Toc28505)

[设置代理 31](#_Toc32437)

# 

# 序

用户imoc

密码072870434

ssh协议需要一个服务器和一个客户端

创建用户useradd imooc-user1

设置密码passwd imooc-user1

# 修改网络配置

修改网络配置：vi /etc/sysconfig/network-scripts/ifcfg-ens33

修改ONBOOT=yes

重启：/etc/init.d/network restart

查看ip地址：ip addr

# 使用yum进行软件包管理：

安装：yum install <package\_name>

更新：yum update <package\_name>

列出所有可安装软件清单：yum list

删除：yum remove <package\_name>

安装ifconfig

yum -y install net-tools

## 安装ssh

安装所有ssh包：yum -y install openssh\*

## 安装lrzsz通信套件

yum -y install lrzsz

## 安装python基础环境和python

yum -y install gcc\*

yum install zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel readline-devel tk-devel sqlite zlib readline openssl libffi libffi-devel

上传python

解压 tar -xvf python3.7

-x：解压 -v：详细输出 -f：必输，指定文件

进入解压后文件输入 ./configure --prefix="安装路径" --with-ssl

制作make文件：make

make install

添加环境变量：vi .bashrc

最后添加alias python3="/home/imooc-user1/python3/bin/python3"

alias pip3="/home/imooc-user1/python3/bin/pip3"

重启环境变量：source .bashrc

# VIM

vim 一个不存在的文件，就相当于新建的

i：在光标前面插入文字 a：在光标后面插入文字

o：在下一行插入文字 O：在上一行插入文字

I：在行首插入文字 A：在行尾插入文字

x:删除后面的字符 X:删除前面的字符

dd：剪切一行（删除） 4dd：剪切4行（删除）

yy：复制一行 4yy：复制4行

gg：到开头 G：到末尾

p：在当前行的下面粘贴 P：在当前行的上面粘贴

u：撤回 .：重复上一次操作

r：替换单个字符

V：选取

shift+>:向右移动

shift+<:向左移动

/：查找

# 安装图形桌面

安装基础环境：

yum -y groupinstall "X Window System"

安装桌面：

yum -y groupinstall "GNOME Desktop"

安装完后，reboot重启

设置图形启动：

systemctl set-default graphical.target

然后在重启reboot

# 安装mongodb

## 方式一：源码安装方式

<https://www.mongodb.com/try/download/community>

下载linux版的mongodb，传到linux上去（rz）

解压缩tar -xvzf mongodb-linux-x86\_64-4.0.9.tgz

解压缩后移动文件：mv mongodb-linux-x86\_64-4.0.9 到 /usr/local/mongodb

家目录，vim .bashrc配置环境变量export PATH=/usr/local/mongodb/bin:$PATH

生效环境变量source .bashrc

创建mkdir mongodb\_dir，cd进去

创建存储数据的目录mkdir data，创建存储log的目录mkdir log

启动：mongod --dbpath data/ --logpath log/mongo.log --port 27017 --fork

dbpath 指定数据路径在data

logpath 指定log存放在log/mongo.log

port 端口号

fork 以后台静默的方式运行的

输mongo测试

## 方式二：yum包管理工具安装方式

vim /etc/yum.repos.d/mongo.repo

然后插入：

[mongodb-org-4.0]

name=MongoDB Repository

baseurl=https://repo.mongodb.org/yum/7Server/mongodb-org/4.0/x86\_64/

gpgcheck=0

enabled=1

退出，yum -y install mongodb-org安装

修改配置：vim /etc/mongod.conf

修改bind为0.0.0.0

启动mongodb:

systemctl enable mongod

systemctl start mongod

查看状态：systemctl status mongod

# Request

文档：<https://requests.readthedocs.io/zh_CN/latest/>

## request模块支持的http方法

1、GET：当客户端向Web服务器请求一个资源的时候使用它被用来访问静态资源，比如HTML文档和图片等。

2、POST：当客户端向服务器端发送信息或者数据的时候使用表单提交（向Web服务器发送大量的复杂的数据）

1. HEAD：当客户端向Web服务器请求一个资源的一些信息而不是资源的全部信息的时候使用。

主要用于确认URL的有效性以及资源更新的日期时间等。

4、PUT：客户端向Web服务器端指定URL发送一个替换的文档或者上传一个新文档的时候使用

5、DELETE：当客户端尝试从Web服务端删除一个由请求URL唯一标识的文档的时候使用。

6、TRACE：当客户端要求可用的代理服务器或者中间服务更改请求来宣布自己的时候使用。

7、OPTIONS：当客户端想要决定其他可用的方法来检测或者处理Web服务器端的一个文档时使用。

8、CONNECT：当客户端想要确定一个明确的连接到远程主机的时候使用，通常便于通过Http代理服务器进行SSL加密通信（Https）连接使用

## url构造方法

import requests

data = {'key1': 'value1', 'key2': 'value2'}

response = requests.get('http://httpbin.org/get', **params**=data)

查看返回头（响应头）：response.headers

查看请求体：response.text

response.text 返回的是一个 unicode 型的文本数据，response.content 返回的 是bytes 型的二进制数据

文本数据可以通过response.text

图片，文件则通过response.content

json数据：response.**json()**

查看状态码：response.status\_code

## 常见状态码

**2xx （3种）**

200 OK：表示从客户端发送给服务器的请求被正常处理并返回；

204 No Content：表示客户端发送给客户端的请求得到了成功处理，但在返回的响应报文中不含实体的主体部分（没有资源可以返回）；

206 Patial Content：表示客户端进行了范围请求，并且服务器成功执行了这部分的GET请求，响应报文中包含由Content-Range指定范围的实体内容。

**3xx （5种）**

301 Moved Permanently：永久性重定向，表示请求的资源被分配了新的URL，之后应使用更改的URL；

302 Found：临时性重定向，表示请求的资源被分配了新的URL，希望本次访问使用新的URL；

 301与302的区别：前者是永久移动，后者是临时移动（之后可能还会更改URL）

303 See Other：表示请求的资源被分配了新的URL，应使用GET方法定向获取请求的资源；

 302与303的区别：后者明确表示客户端应当采用GET方式获取资源

304 Not Modified：表示客户端发送附带条件（是指采用GET方法的请求报文中包含if-Match、If-Modified-Since、If-None-Match、If-Range、If-Unmodified-Since中任一首部）的请求时，服务器端允许访问资源，但是请求为满足条件的情况下返回改状态码；

307 Temporary Redirect：临时重定向，与303有着相同的含义，307会遵照浏览器标准不会从POST变成GET；（不同浏览器可能会出现不同的情况）；

**4xx （4种）**

400 Bad Request：表示请求报文中存在语法错误；

401 Unauthorized：未经许可，需要通过HTTP认证；

403 Forbidden：服务器拒绝该次访问（访问权限出现问题）

404 Not Found：表示服务器上无法找到请求的资源，除此之外，也可以在服务器拒绝请求但不想给拒绝原因时使用；

**5xx （2种）**

500 Inter Server Error：表示服务器在执行请求时发生了错误，也有可能是web应用存在的bug或某些临时的错误时；

503 Server Unavailable：表示服务器暂时处于超负载或正在进行停机维护，无法处理请求；

查看请求头：response.request.headers

## 重定义请求头

header={  
 'User-Agent': 'python-requests/2.24.0'  
}  
response = requests.get('http://httpbin.org/ip',headers=header)

## 设置超时时间

response = requests.get('http://httpbin.org/ip',timeout=1)

## 设置cookies

url = 'http://httpbin.org/cookies'  
cookies = dict(cookies\_are='hello imooc')  
response = requests.get(url=url, cookies=cookies)

## 保持登录session

import requests  
import re  
import time  
  
index\_url = 'http://account.chinaunix.net/login'  
header={  
 请求头参数  
}  
login\_session=requests.session()  
token\_search=re.compile(r'name="\_token"\svalue="(.\*)">')  
index\_response=login\_session.get(url=index\_url,headers=header)  
token\_value=re.search(token\_search,index\_response.text).group(1)  
data={

#post请求传输的数据  
'username': 'dazhuang\_imooc',  
'password': 'abcd1234',  
'\_token':token\_value,  
'\_t':int(time.time())  
}  
login\_url='http://account.chinaunix.net/login/login'  
login\_response=login\_session.post(url=login\_url,headers=header,data=data)  
  
phone\_url='http://account.chinaunix.net/ucenter/user/index'  
phone\_response=login\_session.get(url=phone\_url,headers=header)  
print(phone\_response.text)

## 设置代理

proxy = {

'http': 'http://HY75ZB01X0QZX39D:AB3FB63F433F3F00@http-dyn.abuyun.com:9020',

'https': 'http://HY75ZB01X0QZX39D:AB3FB63F433F3F00@http-dyn.abuyun.com:9020',

}

response = requests.get(url=url, **proxies**=proxy)

## 证书校验

SSL验证默认是开启的，如果证书验证失败，Requests会抛出SSLError异常

1、可以把证书关闭

2、可以加上证书的地址，已经下载好的

response = requests.get(url=url, verify=False)

# re模块

complie函数

complie函数用于编译正则表达式，生成一个Pattern对象

**pattern = re.compile(r'\d+')**

**pattern=re.complie(pattern[,flag])**

pattern：是一个字符串形式的正则表达式

flag是一个可选参数，表示匹配模式，可以使用’|’表示同时生效

****1).re.I(re.IGNORECASE): 忽略大小写   
 2).re.M(MULTILINE): 多行模式，改变'^'和'$'的行为   
 3).re.S(DOTALL): 点任意匹配模式，改变'.'的行为   
 4).re.L(LOCALE): 使预定字符类 \w \W \b \B \s \S 取决于当前区域设定   
 5).re.U(UNICODE): 使预定字符类 \w \W \b \B \s \S \d \D 取决于unicode定义的字符属性   
 6).re.X(VERBOSE): 详细模式。这个模式下正则表达式可以是多行，忽略空白字符，并可以加入注释****

match方法

match方法从头部开始只匹配一次（可以指定起始位置）**pattern.match(string[,pos[endpos]])**

string：是要匹配的字符串

pos和endpos：是可选参数，指定字符串的起始和终点位置

## search方法

search方法查找字符串的任何位置，只匹配一次

**pattern.search(string[,pos[endpos]])**

string：是要匹配的字符串

pos和endpos：是可选参数，指定字符串的起始和终点位置

## findall方法

findall方法用于搜索整个字符串，返回列表

**pattern.findall(string[,pos[endpos]])**

string：是要匹配的字符串

pos和endpos：是可选参数，指定字符串的起始和终点位置

## split方法

split方法按照能够匹配的子串将字符串分隔后返回列表

**pattern.split(string[,maxsplit])**

string：是要匹配的字符串

maxsplit：是可选参数，最多匹配次数

pattern = re.compile(r'[\s\,\;]+')

string = 'a,b;;c d'

print(pattern.split(string))

## sub方法

sub方法用于替换

**pattern.sub(repl,string[,count])**

repl：可以是字符串也可以是一个函数

string：是要匹配的字符串

**count**：是可选参数，匹配次数

例一：字符串

string = '<h1 class="test">imooc</h1>'  
pattern = re.compile('\d')  
print(pattern.sub('2', string))

例二：

string = '<h1 class="test">imooc</h1>'  
pattern = re.compile('<(.\d)\sclass="(.\*?)">.\*?</(\\1)>')  
print(pattern.search(string).group(1))

例三：函数

string = '<h1 class="test">imooc</h1>'  
pattern = re.compile('<(.\d)\sclass="(.\*?)">.\*?</(\\1)>')  
def func(m):  
 return 'after sub'+m.group(2)  
print(pattern.sub(func,string))

# xpath语法

/在根节点上进行选取，//从任意位置进行选取

//div[@class='cont\_sourece']/text()

一个点代表选取当前节点，两个点代表选取当前节点的父节点

//div[@class='cont\_sourece']/.

//div[@class='cont\_sourece']/..

模糊查询，获取所有包含content属性的div节点

//div[contains(@class,'content')]

取反

//div[not(contains(@class,'content'))]

取倒数第13个标签

//div[@class='contains']/p[last()-13]

获取p节点大于61的p节点

//div[@class='contains']/p[position()>61]

# BeautifulSoup4

**prettify()**：标准格式化

from bs4 import BeautifulSoup  
soup=BeautifulSoup(html,'lxml')  
print(soup.prettify())

**attrs**：返回标签所有属性值（字典）

print(soup.p.attrs)

**contents：**获取子节点，返回列表

print(soup.p.contents)

**children：**同contents，但返回迭代器，要用for遍历

print(soup.p.children)

**descendants**：获取子节点和孙节点，返回迭代器

print(soup.p.descendants)

**parent**：获取父节点

print(soup.a.parent)

**parents**：获取所有祖先节点，返回迭代器

print(soup.a.parents)

**next\_sibling**：获取下一个兄弟节点

**previous\_sibling**：获取上一个兄弟节点

print(soup.a.next\_sibling)

print(soup.a.previous\_sibling)

**next\_siblings**：获取之后的兄弟节点

**previous\_siblings**：获取之前的兄弟节点

print(soup.a.next\_siblings)

print(soup.a.previous\_siblings)

## 方法选择器

find\_all：通过节点名查询

print(soup.find\_all(name='p')[0])

或

for ul in soup.find\_all(name='ul'):  
 print(ul.find\_all(name='li'))

通过属性查询

print(soup.find\_all(attrs={'id':'list1'}))

## css选择器

print(soup.select('.panel .panel\_heading'))

# lxml

from lxml import etree  
html = etree.HTML(html\_data)  
print(etree.tostring(html).decode())

result=html.xpath('//li/a/text()')

# 案例：lxml爬取51招聘网

import requests  
from lxml import etree  
import json  
  
url='https://search.51job.com/list/030000,000000,0000,00,9,99,python,2,1.html'  
header={  
 'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9',  
 'Accept-Encoding': 'gzip, deflate, br',  
 'Accept-Language': 'zh-CN,zh;q=0.9',  
 'Connection': 'keep-alive',  
 'Host': 'search.51job.com',  
 'Upgrade-Insecure-Requests': '1',  
 'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.116 Safari/537.36'  
}  
response=requests.get(url=url,headers=header)  
response.encoding='gbk'  
html = etree.HTML(response.text)  
result=html.xpath('//div[@class="dw\_table"]/div[@class="el"]')  
list=[]  
for i in result:  
 info={}  
 info['job\_name']=i.xpath('./p/span/a/@title')[0]  
 info['company\_name'] = i.xpath('./span/a/@title')[0]  
 info['company\_address'] = i.xpath('./span[@class="t3"]/text()')[0]  
 info['money'] = i.xpath('./span[@class="t4"]/text()')[0]  
 info['date'] = i.xpath('./span[@class="t5"]/text()')[0]  
 list.append(info)  
print(json.dumps(list))

## 爬取数据并保存到mongo

import requests  
from lxml import etree  
import pymongo  
  
myclient=pymongo.MongoClient("mongodb://192.168.2.6:27017")  
mydb=myclient['db\_51job']  
mycollection=mydb['collection\_51job']  
  
url='https://search.51job.com/list/030000,000000,0000,00,9,99,python,2,1.html'  
header={  
 'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9',  
 'Accept-Encoding': 'gzip, deflate, br',  
 'Accept-Language': 'zh-CN,zh;q=0.9',  
 'Connection': 'keep-alive',  
 'Host': 'search.51job.com',  
 'Upgrade-Insecure-Requests': '1',  
 'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.116 Safari/537.36'  
}  
response=requests.get(url=url,headers=header)  
response.encoding='gbk'  
html = etree.HTML(response.text)  
result=html.xpath('//div[@class="dw\_table"]/div[@class="el"]')  
list=[]  
for i in result:  
 info={}  
 info['job\_name']=i.xpath('./p/span/a/@title')[0]  
 info['company\_name'] = i.xpath('./span/a/@title')[0]  
 info['company\_address'] = i.xpath('./span[@class="t3"]/text()')[0]  
 #处理无数据报错  
 try:  
 info['money'] = i.xpath('./span[@class="t4"]/text()')[0]  
 except IndexError:  
 info['money'] = '无数据'  
 info['date'] = i.xpath('./span[@class="t5"]/text()')[0]  
 list.append(info)  
  
# mycollection.insert\_many(list)  
result=mycollection.find()  
for item in result:  
 print(item)

# 案例：爬取51招聘网全部网页信息

## handle\_51job.py

import requests  
from multiprocessing import Queue  
from lxml import etree  
import threading  
from handle\_mongo import insert\_data  
  
  
class Crawl\_page(threading.Thread):  
 def \_\_init\_\_(self, thread\_name, page\_queue, data\_queue):  
 super(Crawl\_page, self).\_\_init\_\_()  
 # 线程名称  
 self.thread\_name = thread\_name  
 self.page\_queue = page\_queue  
 self.data\_queue = data\_queue  
 # 默认请求头  
 self.header = {  
 'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9',  
 'Accept-Encoding': 'gzip, deflate, br',  
 'Accept-Language': 'zh-CN,zh;q=0.9',  
 'Connection': 'keep-alive',  
 'Host': 'search.51job.com',  
 'Upgrade-Insecure-Requests': '1',  
 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.116 Safari/537.36'  
 }  
  
 def run(self):  
 print('当前为:{}'.format(self.thread\_name))  
 while not page\_flag:  
 try:  
 page = self.page\_queue.get(block=False)  
 page\_url = 'https://search.51job.com/list/030000,000000,0000,00,9,99,python,2,' + str(page) + '.html'  
 print('当前网址是：{}'.format(page\_url))  
 response = requests.get(url=page\_url, headers=self.header)  
 response.encoding = 'gbk'  
 self.data\_queue.put(response.text)  
 except:  
 pass  
  
  
class Crawl\_html(threading.Thread):  
 def \_\_init\_\_(self, thread\_name, data\_queue, lock):  
 super(Crawl\_html, self).\_\_init\_\_()  
 self.thread\_name = thread\_name  
 self.data\_queue = data\_queue  
 self.lock = lock  
  
 def parse(self, text):  
 html = etree.HTML(text)  
 result = html.xpath('//div[@class="dw\_table"]/div[@class="el"]')  
 list = []  
 for i in result:  
 info = {}  
 info['job\_name'] = i.xpath('./p/span/a/@title')[0]  
 info['company\_name'] = i.xpath('./span/a/@title')[0]  
 info['company\_address'] = i.xpath('./span[@class="t3"]/text()')[0]  
 # 处理无数据报错  
 try:  
 info['money'] = i.xpath('./span[@class="t4"]/text()')[0]  
 except IndexError:  
 info['money'] = '无数据'  
 info['date'] = i.xpath('./span[@class="t5"]/text()')[0]  
 list.append(info)  
 return list  
  
 def run(self):  
 print('当前文本线程是：{}'.format(self.thread\_name))  
 while not data\_flag:  
 try:  
 text = self.data\_queue.get(block=False)  
 result = self.parse(text)  
 with self.lock:  
 insert\_data.insert\_db(result)  
 except:  
 pass  
  
  
page\_flag = False  
data\_flag = False  
  
  
def main():  
 # 定义两个队列，放页码和文本  
 page\_queue = Queue()  
 data\_queue = Queue()  
 lock = threading.Lock()  
  
 # 将页码加入队列  
 for page in range(1, 93):  
 page\_queue.put(page)  
  
 print('一共有{}页'.format(page\_queue.qsize()))  
  
 # 处理页码线程  
 crawl\_page\_list = ['页码线程1号', '页码线程2号', '页码线程3号']  
 page\_thread\_list = []  
 for thread\_name\_page in crawl\_page\_list:  
 thread\_page = Crawl\_page(thread\_name\_page, page\_queue, data\_queue)  
 thread\_page.start()  
 page\_thread\_list.append(thread\_page)  
  
 # 处理文本线程  
 parse\_list = ['文本线程1号', '文本线程2号', '文本线程3号']  
 parse\_thread\_list = []  
 for thread\_name\_parse in parse\_list:  
 thread\_parse = Crawl\_html(thread\_name\_parse, data\_queue, lock)  
 thread\_parse.start()  
 parse\_thread\_list.append(thread\_parse)  
  
 # 结束页码机制  
 global page\_flag  
 while not page\_queue.empty():  
 pass  
 page\_flag = True  
 for thread\_page\_join in page\_thread\_list:  
 thread\_page\_join.join()  
 print(thread\_page\_join.thread\_name, 'OK!')  
  
 # 结束文本机制  
 global data\_flag  
 while not data\_queue.empty():  
 pass  
 data\_flag = True  
 for thread\_parse\_join in parse\_thread\_list:  
 thread\_parse\_join.join()  
 print(thread\_parse\_join.thread\_name, 'OK!')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()  
 result=insert\_data.find\_db()  
 for i in result:  
 print(i)

## handle\_mongo.py

import pymongo  
  
  
class Mongo\_client(object):  
 def \_\_init\_\_(self):  
 myclient = pymongo.MongoClient("mongodb://192.168.1.102:27017")  
 mydb = myclient['db\_51job']  
 self.mycollection = mydb['collection\_51job']  
  
 def insert\_db(self, item):  
 self.mycollection.insert\_many(item)  
  
 def find\_db(self):  
 return self.mycollection.find()  
  
insert\_data = Mongo\_client()

## 设置代理

def run(self):  
 print('当前为:{}'.format(self.thread\_name))  
 while not page\_flag:  
 try:  
 page = self.page\_queue.get(block=False)  
 page\_url = 'https://search.51job.com/list/030000,000000,0000,00,9,99,python,2,' + str(page) + '.html'  
 print('当前网址是：{}'.format(page\_url))  
 #设置阿布云代理  
  **proxy = {  
 'http': 'http://HY75ZB01X0QZX39D:AB3FB63F433F3F00@http-dyn.abuyun.com:9020',  
 'https': 'http://HY75ZB01X0QZX39D:AB3FB63F433F3F00@http-dyn.abuyun.com:9020',  
 }  
 response = requests.get(url=page\_url, headers=self.header,proxies=proxy)** response.encoding = 'gbk'  
 self.data\_queue.put(response.text)  
 except:  
 pass

# 实操案例爬取豆瓣T250

## crawl\_douban\_movie\_info\_top250.py

import re  
import requests  
from lxml import etree  
from concurrent.futures import ThreadPoolExecutor  
from handle\_mongo import douban\_mongo  
  
  
class HandleDoubanMovieTop250(object):  
 def \_\_init\_\_(self):  
 # 定义请求头  
 self.header = {  
 'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9',  
 'Accept-Language': 'zh-CN,zh;q=0.9',  
 'Cache-Control': 'max-age=0',  
 'Connection': 'keep-alive',  
 'Host': 'movie.douban.com',  
 'Upgrade-Insecure-Requests': '1',  
 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.116 Safari/537.36'  
 }  
 # 存放地址列表  
 self.page\_url = []  
  
 # 定义页码处理方法  
 def handle\_page\_url(self):  
 for i in range(10):  
 url = 'https://movie.douban.com/top250?start={}'.format(i \* 25)  
 self.page\_url.append(url)  
  
 # 定义请求网页方法  
 def handle\_request(self, url):  
 response = requests.get(url=url, headers=self.header)  
 return response.text  
  
 # 定义处理网页文本方法  
 def handle\_page\_detail(self, url):  
 text = self.handle\_request(url)  
 # 用正则处理网页，去除空格，换行符  
 pattern = re.compile(r'&nbsp;|\n| ')  
 text = pattern.sub('', text)  
 # 格式化网页文本  
 html = etree.HTML(text)  
 result = html.xpath('//div[@class="article"]//div[@class="info"]')  
 list = []  
 for i in result:  
 # 提取网页数据  
 info = {}  
 info['movie\_name'] = self.join(i.xpath('.//a/span/text()'))  
 info['actors\_information'] = self.join(i.xpath('./div[@class="bd"]/p/text()'))  
 info['score'] = i.xpath('.//span[@class="rating\_num"]/text()')  
 info['evaluate'] = i.xpath('.//span[4]/text()')  
 info['describe'] = i.xpath('.//span[@class="inq"]/text()')  
 info['from\_url'] = url  
 list.append(info)  
 # 把数据存入数据库  
 douban\_mongo.handle\_save\_data(list)  
  
 # 拼接字符串方法  
 def join(self, list):  
 string = ''  
 for i in list:  
 string += str(i)  
 return string  
  
 def run(self):  
 self.handle\_page\_url()  
 # 用线程池方法运行  
 executor = ThreadPoolExecutor()  
 for url in self.page\_url:  
 executor.submit(self.handle\_page\_detail, url)  
 executor.shutdown()  
  
  
# 程序入口函数  
def main():  
 douban = HandleDoubanMovieTop250()  
 douban.run()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

## handle\_mongo.py

import pymongo  
  
  
class Handle\_Mongo(object):  
 def \_\_init\_\_(self):  
 # 连接mongo数据库  
 myclient = pymongo.MongoClient("mongodb://192.168.1.106:27017")  
 mydb = myclient['douban']  
 self.mycollection = mydb['douban\_data']  
  
 # 定义数据插入方法  
 def handle\_save\_data(self, item):  
 self.mycollection.insert\_many(item)  
  
  
douban\_mongo = Handle\_Mongo()

# PyMongo库的使用

import pymongo  
  
myclient = pymongo.MongoClient("mongodb://192.168.2.6:27017")  
mydb = myclient['imoc']  
mycollection = mydb['pymongo\_test']  
# 插入一条记录  
mycollection.insert\_one({"name": "imooc", "flag": 1, "url": "https://www.imooc.com"})  
mylist = [  
 {"name": "taobao", "flag": 100, "url": "https://www.taobao.com"},  
 {"name": "qq", "flag": 101, "url": "http://www.qq.com"},  
]  
# 插入多条数据，  
mycollection.insert\_many(mylist)  
  
# 查看find方法，返回游标  
result = mycollection.find()  
for item in result:  
 print(item)  
# 0不显示，1显示，第一个的{}表示查找所有  
result = mycollection.find({}, {'\_id': 0, "name": 1, "flag": 1})  
# find\_one查看第一条数据,返回数据，不是游标  
result = mycollection.find\_one()  
# find高级查询，大于$gt小于$lt,返回游标  
result = mycollection.find({"flag": {"$gt": 100}})  
# find正则表达式

result = mycollection.find({"name": {"$regex": "^G"}})

# update\_one更改数据，改第一条;第一个old，第二个new  
mycollection.update\_one({"name":{"$regex":"^G"}},{"$set":{"name":"jd"}})  
#uodate\_many更改多条数据,{}表示查找所有  
mycollection.update\_many({},{"$set":{"name":"imooc"}})  
  
# delete\_one删除一条数据  
mycollection.delete\_one({"url":"https://www.imooc.com"})  
#delete\_many删除多条数据  
mycollection.delete\_many({"url":{"$regex":"https?://www\.[tq]"}})  
#删除所有数据，{}代表所有数据  
mycollection.delete\_many({})

## 报错

pymongo.errors.ServerSelectionTimeoutError: xxxx: timed out

方法：关闭防火墙，加上--bind\_ip 0.0.0.0

关闭防火墙：  
临时关闭防火墙：systemctl stop firewalld.service  
永久关闭防火墙：systemctl disable firewalld.service开关机自启动设置  
启动防火墙systemctl start firewalld.service  
永久启动防火墙systemctl enable firewalld开关机自启动设置  
重启mongo服务：mongod --dbpath data/ --logpath log/mongo.log --port 27017 --bind\_ip 0.0.0.0 --fork

# selenium模块

浏览器驱动driver下载地址

Phantomjs：http://phantomjs.org/download.html

Mozilla Firefox：http://github.com/mozilla/geckodriver/releases

Google Chrome：http://chromedriver.storage.googleapis.com/index.html

注意：

下载驱动后，要添加到环境变量中

--version查看版本

Google Chrome驱动，需要对应浏览器的版本

import time  
from selenium import webdriver  
  
  
test\_webdriver=webdriver.Chrome(executable\_path="D:\webdriver\chromedriver.exe")  
test\_webdriver.get('https://echarts.apache.org/examples/zh/index.html')  
time.sleep(5)  
test\_webdriver.quit()

通过xpath提取星系

for item in test\_webdriver.find\_elements\_by\_xpath('//h4[@class="chart-title"]'):  
 print(item.text)

发送信息

test\_webdriver.find\_element\_by\_xpath('//input[@id="kw"]').send\_keys('python')

点击

test\_webdriver.find\_element\_by\_xpath('//input[@id="su"]').click()

窗口大小

test\_webdriver.set\_window\_size(480, 800)

控制浏览器前进、后退、刷新：

后退：test\_webdriver.back()

前进：test\_webdriver.forward()

刷新：est\_webdriver.refresh()

[响应鼠标事件](https://class.imooc.com/lesson/1115" \l "mid=26431)

selenium控制鼠标：

perform()：执行所有ActionChains中存储的行为；

context\_click()：右击

double\_click()：双击

drag\_and\_drop()：拖动

move\_to\_element()：鼠标悬停

import time  
from selenium import webdriver  
from selenium.webdriver.common.action\_chains import ActionChains

test\_driver=webdriver.Chrome(executable\_path="D:\webdriver\chromedriver.exe")  
test\_driver.maximize\_window()  
test\_driver.get("https://www.baidu.com")  
above=test\_driver.find\_element\_by\_xpath('//\*[@id="s-usersetting-top"]')  
ActionChains(test\_driver).move\_to\_element(above).perform()  
time.sleep(3)  
test\_driver.quit()

# selenium元素定位技巧

selenium提供了8中定位方式：

find\_element\_by\_id()

find\_element\_by\_name()

find\_element\_by\_class\_name()

find\_element\_by\_tag\_name()

find\_element\_by\_link\_text()

find\_element\_by\_partial\_link\_text()

find\_element\_by\_xpath()

find\_element\_by\_selector()

[显示等待和隐式等待](https://class.imooc.com/lesson/1115" \l "mid=26424)

## 显示等待

import time  
from selenium import webdriver  
#简写用包  
from selenium.webdriver.common.by import By  
#等待用包  
from selenium.webdriver.support.ui import WebDriverWait  
#场景判断，判断某元素是否出现  
from selenium.webdriver.support import expected\_conditions as EC  
  
test\_webdriver=webdriver.Chrome(executable\_path="D:\webdriver\chromedriver.exe")  
test\_webdriver.get('https://www.baidu.com')

# 显式等待，传3个参数test\_driver、timeout、轮询时间  
# until等到某一个控件出来，通过EC.presence\_of\_element\_located来找元素  
element=WebDriverWait(test\_webdriver,5,0.5).until(EC.presence\_of\_element\_located((By.ID,'kw')))  
element.send\_keys('python')  
time.sleep(2)  
test\_webdriver.quit()

## 隐式等待

import time  
from selenium import webdriver  
from selenium.common.exceptions import NoSuchAttributeException  
  
test\_driver = webdriver.Chrome(executable\_path="D:\webdriver\chromedriver.exe")  
test\_driver.implicitly\_wait(5)  
test\_driver.get('https://www.baidu.com')  
try:  
 test\_driver.find\_element\_by\_id('1').send\_keys('python')  
 time.sleep(2)  
except NoSuchAttributeException as e:  
 print(e)  
  
test\_driver.quit()

# 案例：Selenium\_51job

import time  
from selenium import webdriver  
# 简写用包  
from selenium.webdriver.common.by import By  
# 等待用包  
from selenium.webdriver.support.ui import WebDriverWait  
# 场景判断，判断某元素是否出现  
from selenium.webdriver.support import expected\_conditions as EC  
  
  
class Handle\_webdriver(object):  
 def \_\_init\_\_(self):  
 self.driver = webdriver.Chrome()  
 self.driver.maximize\_window()  
  
 def handle\_job(self):  
 self.driver.get('https://search.51job.com/list/000000,000000,0000,00,9,99,%20,2,1.html')  
 if WebDriverWait(self.driver, 5, 0.5).until(EC.presence\_of\_element\_located((By.ID, 'kwdselectid'))):  
 input\_keyword = input('请输入查找的岗位：')  
 self.driver.find\_element\_by\_id('kwdselectid').send\_keys(input\_keyword)  
 self.driver.find\_element\_by\_class\_name('p\_but').click()  
 if WebDriverWait(self.driver, 5, 0.5).until(EC.presence\_of\_element\_located((By.ID, 'resultList'))):  
 while True:  
 # 处理网页代码  
 # 处理网页代码  
 # 处理网页代码  
 time.sleep(2)  
 if self.driver.find\_element\_by\_xpath('//li[@class="bk"][2]/a').text=='下一页':  
 self.driver.find\_element\_by\_xpath('//li[@class="bk"][2]/a').click()  
 else:  
 break  
 self.driver.quit()  
  
  
text\_selenium = Handle\_webdriver()  
text\_selenium.handle\_job()

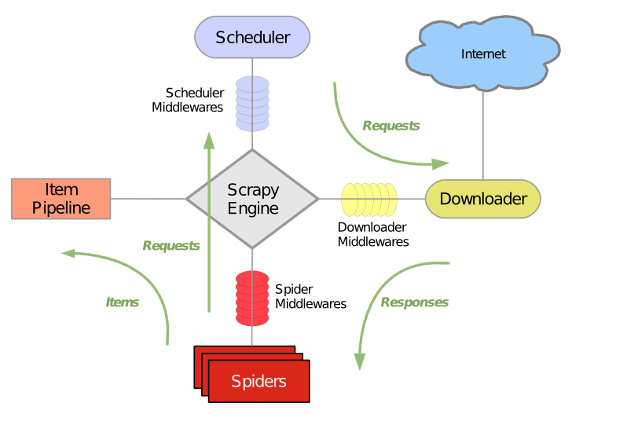
## 报错

错误：依赖检测失败：

方法：yum localinstall 安装包依赖

# scrapy 框架

## Scrapy运行流程

1. 引擎（scrapy）从调度器（scheduler）中取出一个链接(URL)用于接下来的抓取
2. 引擎把URL封装成一个请求(Request)传给下载器（downloader）
3. 下载器（downloader）把资源下载下来，并封装成应答包(Response)
4. 爬虫（spider）解析Response
5. 解析出实体（Item）,则交给实体管道（item-pipelines）进行进一步的处理
6. 解析出的是链接（URL）,则把URL交给调度器（scheduler）等待抓取

Scrapy 框架中 有8 个主要组件：

1. ****Scrapy引擎(Scrapy engine)****：这是框架的核心，用来处理整个系统的数据流处理，触发事务；
2. ****调度器(Scheduler)****：接受引擎发来的请求，压入队列中，并在引擎再次请求的时候返回。这个调度器决定URL的优先队列，并且做了去重处理；
3. ****下载器(Downloader)****：用于下载网页内容, 并将网页内容返回给爬虫(Scrapy下载器是建立在twisted这个高效的异步模型上的)；
4. ****爬虫(Spiders)****：用于从特定的网页中提取自己需要的信息, 即所谓的实体(Item)。用户也可以从中提取出链接，让Scrapy引擎继续抓取下一个页面；
5. ****项目管道(Pipeline)****：负责处理爬虫从网页中抽取的实体，主要的功能是持久化实体、验证实体的有效性、清除不需要的信息。当页面被爬虫解析后，将被发送到项目管道，并经过几个特定的次序处理数据；
6. ****下载器中间件(Downloader Middlewares)****：位于Scrapy引擎和下载器之间的中间件，主要是处理Scrapy引擎与下载器之间的请求及响应；
7. ****爬虫中间件(Spider Middlewares)****：介于Scrapy引擎和爬虫之间的中间件，主要工作是处理爬虫的响应输入和请求输出；
8. ****调度中间件(Scheduler Middewares)****：介于Scrapy引擎和调度器之间的中间件，从Scrapy引擎发送到调度的请求和响应。

**新建scrapy项目**：**scrapy startproject tubatu\_scrapy\_project**

scrapy.cfg项目的配置文件

items.py：定义items数据结构的地方（想要抓取哪些字段，比如名称、url）

pipeline.py：数据存储json、mongo文件的时候，可以编写逻辑

settings.py：项目的设置文件（设置user\_agent、是否遵循robots协议等）

**设置ROBOTSTXT\_OBEY = False**

spiders：每个爬虫项目的实现，解析规则会写到这里面

middlewares.py：有两个中间件类

**生成spiders文件：（通过模板进行生成爬虫文件）**

在爬虫项目的spiders文件中

scrapy genspider 项目名称URL

**scrapy genspider tubatu xiaoguotu.to8to.com/tuce/**

cmd**运行scrapy项目**：**scrapy crawl tubatu**

可通过：

from scrapy import cmdline  
cmdline.execute('scrapy crawl tubatu --nolog'.split())

## 设置settings参数

#请求速度  
DOWNLOAD\_DELAY = 0.2  
#超时时间  
DOWNLOAD\_TIMEOUT=10  
#是否重试  
RETRY\_ENABLED=True  
#重试次数  
RETRY\_TIMES=3

## items.py

import scrapy  
  
  
class TubatuScarpyProjectItem(scrapy.Item):  
 # define the fields for your item here like:  
 # name = scrapy.Field()  
 # 装修名称  
 content\_name = scrapy.Field()  
 # 请求id  
 content\_id = scrapy.Field()  
 # 请求url  
 content\_url = scrapy.Field()  
 # 昵称  
 nick\_name = scrapy.Field()  
 # 图片url  
 pic\_url = scrapy.Field()  
 # 图片名称  
 pic\_name = scrapy.Field()

## tubatu.py

import scrapy  
import re  
import json  
from tubatu\_scarpy\_project.items import TubatuScarpyProjectItem  
  
  
class TubatuSpider(scrapy.Spider):  
 #名称不能冲突，也就是说不能重复,name不能冲突  
 name = 'tubatu'  
 #允许爬虫去抓取的域名  
 allowed\_domains = ['xiaoguotu.to8to.com']  
 #项目启动之后要启动的爬虫文件  
 start\_urls = ['https://xiaoguotu.to8to.com/tuce/p\_1.html']  
  
 #默认的解析方法  
 def parse(self, response):  
 #这里使用了正则表达式来获取项目的id,需要使用转义字符来转义这个.  
 content\_id\_search = re.compile(r"(\d+)\.html")  
 #response后面可以直接使用xpath方法  
 #response就是一个Html对象  
 pic\_item\_list = response.xpath("//div[@class='item']")  
 for item in pic\_item\_list:  
 info = {}  
 #这里有一个点不要丢了,是说明在当前的Item下面再次使用xpath  
 #我们可以通过extract\_first这个方法来获取项目的名称,项目的数据  
 #获取的项目的名称  
 info['content\_name'] = item.xpath(".//div/a/text()").extract\_first()  
 #项目的URL  
 content\_url = 'https:'+item.xpath(".//div/a/@href").extract\_first()  
 info['content\_id'] = content\_id\_search.search(content\_url).group(1)  
 info['content\_ajax\_url'] = 'https://xiaoguotu.to8to.com/case/list?a2=0&a12=&a11='+str(info['content\_id'])+'&a1=0&a17=1'  
 #我们使用yield来发送这个异步请求  
 #使用的是scrapy.Request发送请求的  
 #回调函数,只写方法的名称，不要调用方法  
 yield scrapy.Request(url=info['content\_ajax\_url'],callback=self.handle\_pic\_parse,meta=info)  
  
 #页码逻辑  
 if response.xpath("//a[@id='nextpageid']"):  
 now\_page = int(response.xpath("//div[@class='pages']/strong/text()").extract\_first())  
 next\_page\_url = 'https://xiaoguotu.to8to.com/tuce/p\_%d.html'%(now\_page+1)  
 yield scrapy.Request(url=next\_page\_url,callback=self.parse)  
  
 def handle\_pic\_parse(self,response):  
 pic\_dict\_data = json.loads(response.text)['dataImg']  
 for pic\_item in pic\_dict\_data:  
 for item in pic\_item['album']:  
 tubatu\_info = TubatuScarpyProjectItem()  
 #昵称  
 tubatu\_info['nick\_name'] = item['l']['n']  
 #图片的地址  
 # tubatu\_info['pic\_url'] = 'https://pic1.to8to.com/case/'+item['l']['s']  
 #必须要使用这个字段，后面的数据要改成列表格式  
 tubatu\_info['image\_urls'] = ['https://pic1.to8to.com/case/'+item['l']['s']]  
 #图片的名称  
 tubatu\_info['pic\_name'] = item['l']['t']  
 tubatu\_info['content\_name'] = response.meta['content\_name']  
 tubatu\_info['content\_id'] = response.meta['content\_id']  
 tubatu\_info['content\_url'] = response.meta['content\_ajax\_url']  
 #yield到pipelines，要setting里面启用ITEM\_PIPELINES，如果不启用，是无法使用的  
 yield tubatu\_info

## pipelines.py

import pymongo  
from scrapy.exceptions import DropItem  
from scrapy.pipelines.images import ImagesPipeline  
  
  
class TubatuScarpyProjectPipeline:  
 def \_\_init\_\_(self):  
 myclient = pymongo.MongoClient('mongodb://192.168.1.106:27017')  
 mydb = myclient['db\_tubatu']  
 self.mycollection = mydb['collection\_tubatu']  
  
 def process\_item(self, item, spider):  
 data = dict(item)  
 self.mycollection.insert\_one(data)  
 return item  
  
#自定义图片下载类，继承ImagesPipeline  
class TubatuImagePipeline(ImagesPipeline):  
 def get\_media\_requests(self, item, info):  
 #根据指定的image\_urls进行爬取

#要setting里面启用,并设置保存路径MAGES\_STORE = r'D:\xxx'  
 pass  
  
 def item\_completed(self, results, item, info):  
 #图片下载完成后处理结果  
 image\_paths=[x['path'] for ok,x in results if ok]  
 if not image\_paths:  
 return DropItem('没有图片')  
 return item  
  
  
 def file\_path(self, request, response=None, info=None):  
 #给下载的图片设置名称  
 #要在settings.py中设置IMAGES\_STORE = 'image'  
 url=request.url  
 file\_name=url.split('/')[-1]  
 return file\_name

## middlewares.py

自定义中间件

## 设置随机请求头

#要在settings.py启用中间件 DOWNLOADER\_MIDDLEWARES  
class my\_useragent(object):  
 def process\_request(self, request, spider):  
 #请求头列表  
 user\_agent\_list = [  
 'Mozilla/5.0 (compatible; U; ABrowse 0.6; Syllable) AppleWebKit/420+ (KHTML, like Gecko)',  
 'Mozilla/5.0 (compatible; U; ABrowse 0.6; Syllable) AppleWebKit/420+ (KHTML, like Gecko)',  
 'Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0; Acoo Browser 1.98.744; .NET CLR 3.5.30729)',  
 'Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0; Acoo Browser 1.98.744; .NET CLR 3.5.30729)',  
 'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0; Acoo Browser; GTB5; Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1) ; InfoPath.1; .NET CLR 3.5.30729; .NET CLR 3.0.30618)',  
 'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; SV1; Acoo Browser; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; Avant Browser)',  
 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; Acoo Browser; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.0.04506)',  
 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; Acoo Browser; GTB5; Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1) ; Maxthon; InfoPath.1; .NET CLR 3.5.30729; .NET CLR 3.0.30618)',  
 'Mozilla/4.0 (compatible; Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0; Acoo Browser 1.98.744; .NET CLR 3.5.30729); Windows NT 5.1; Trident/4.0)',  
 'Mozilla/4.0 (compatible; Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; GTB6; Acoo Browser; .NET CLR 1.1.4322; .NET CLR 2.0.50727); Windows NT 5.1; Trident/4.0; Maxthon; .NET CLR 2.0.50727; .NET CLR 1.1.4322; InfoPath.2)',  
 'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0; Acoo Browser; GTB6; Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1) ; InfoPath.1; .NET CLR 3.5.30729; .NET CLR 3.0.30618)'  
 ]  
 #随机选择请求头  
 agent=random.choice(user\_agent\_list)  
 #设置请求头  
 request.headers['User-Agent']=agent

## 设置代理

#要在settings.py启用中间件 DOWNLOADER\_MIDDLEWARES  
class my\_proxy(object):  
 def process\_request(self, request, spider):  
 #proxy,主机头：端口号  
 request.meta['proxy']='代理地址'  
 #用户名：密码  
 proxy\_name\_pass='xxxxxx:xxxxx'.encode('utf-8')  
 encode\_pass\_name=base64.b64encode(proxy\_name\_pass)  
 #将代理设置到头部  
 #注意！！Basic要有空格  
 request.heardes['Proxy-Autorization']='Basic '+encode\_pass\_name.decode()

# 案例：抓取二手车网站汽车数据

[破解反爬JS](https://class.imooc.com/lesson/1102" \l "mid=26460)

import requests  
#通过execjs这个包，来解析js，pip install pyexecjs  
import execjs  
import re  
  
#我们请求城市的接口  
url = 'https://www.guazi.com/www/buy'  
  
#cookie值要删掉，否则对方会根据这个值发现我们，并且屏蔽我们  
#要通过正则表达式处理请求头,里面有空格，大家一定要注意  
header = {  
 "Accept":"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8",  
 "Accept-Encoding":"gzip, deflate, br",  
 "Accept-Language":"zh-CN,zh;q=0.9",  
 "Connection":"keep-alive",  
 "Host":"www.guazi.com",  
 "Referer":"https://www.guazi.com/www/buy",  
 "Upgrade-Insecure-Requests":"1",  
 "User-Agent":"Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3610.2 Safari/537.36",  
}  
  
response = requests.get(url=url,headers=header)  
#设置返回的编码  
response.encoding = 'utf-8'  
if '正在打开中,请稍后' in response.text:  
 #通过正则表达式获取了相关的字段和值  
 value\_search = re.compile(r"anti\('(.\*?)','(.\*?)'\);")  
 string = value\_search.search(response.text).group(1)  
 key = value\_search.search(response.text).group(2)  
 #读取，我们破解的js文件  
 with open('guazi.js','r') as f:  
 f\_read = f.read()  
 #使用execjs包来封装这段JS,传入的是读取后的js文件  
 js = execjs.compile(f\_read)  
 js\_return = js.call('anti',string,key)  
 cookie\_value = 'antipas='+js\_return  
 header['Cookie'] = cookie\_value  
 response\_second = requests.get(url=url,headers=header)  
 print(response\_second.text)

## handle\_guazi\_task.py

import re  
import requests  
import execjs  
import json  
from guazi\_scrapy\_project.handle\_mongo import mongo  
  
url = 'https://www.guazi.com/www/buy/'  
header = {  
 'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9',  
 'Accept-Encoding': 'gzip, deflate, br',  
 'Accept-Language': 'zh-CN,zh;q=0.9',  
 'Connection': 'keep-alive',  
 'Host': 'www.guazi.com',  
 'Upgrade-Insecure-Requests': '1',  
 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.105 Safari/537.36'  
}  
  
response = requests.get(url=url, headers=header)  
response.encoding = 'utf-8'  
if '正在打开中,请稍后' in response.text:  
 value\_search = re.compile(r"anti\('(.\*?)','(.\*?)'\);")  
 string = value\_search.search(response.text).group(1)  
 key = value\_search.search(response.text).group(2)  
 with open('guazi.js', 'r') as f:  
 f\_read = f.read()  
  
 js = execjs.compile(f\_read)  
 js\_return = js.call('anti', string, key)  
 cookie\_value = 'antipas=' + js\_return  
  
 header['Cookie'] = cookie\_value  
 response = requests.get(url=url, headers=header)  
 #以上时破解反爬JS  
  
 #由于二手车网站的城市名js进行了重新编译  
 #用正则找出左列和右列的城市  
 cityLeft\_search = re.compile(r'var\scityLeft\s=\s(.\*);')  
 cityRight\_search = re.compile(r'var\scityRight\s=\s(.\*);')  
 #转成json格式（字典）  
 cityLeft = json.loads(cityLeft\_search.findall(response.text)[0])  
 cityRight = json.loads(cityRight\_search.findall(response.text)[0])  
 #合并两个字典  
 cityAll = dict(cityLeft, \*\*cityRight)  
 city\_list=[]  
 #从字典提取信息  
 for k, v in cityAll.items():  
 for i in v:  
 city\_list.append((i['domain'],i['name']))  
  
 #用正则爬取汽车名称  
 brand\_search=re.compile('href="\/www/(.\*?)\/c-1\/#bread"\s+>(.\*?)</a>')  
 brand\_list=brand\_search.findall(response.text)  
 #制作城市加汽车名的信息  
 for city in city\_list:  
 if city[1]=='北京':  
 for brand in brand\_list:  
 info={}  
 #https://www.guazi.com/bj/buick/i7  
 info['task\_url']='https://www.guazi.com/{}/{}/o1i7'.format(city[0],brand[0])  
 info['city\_name']=city[1]  
 info['brand\_name']=brand[1]  
 mongo.save\_task('guazi\_task',info)